



下载APP



11 | 高性能优化：物理机极致优化

2021-10-20 余志东

《手把手带你搭建秒杀系统》

课程介绍 >



讲述：余志东

时长 13:15 大小 12.14M



你好，我是志东，欢迎和我一起从零打造秒杀系统。

经过前面章节的学习，咱们逻辑代码层面的优化，基本都已经差不多了。这个时候，再制约系统性能的，往往是些逻辑代码之外的因素了，这些我们可能很少接触，但却非常重要。

所以今天我们将学习一下物理机相关的优化思路，以及部署在物理机上的 Nginx 的配置优化。同时说明一下，以下的优化都是在 Linux 平台下的优化方式，在其他平台的话，部分优化可能不支持。



物理机优化

以下优化，在搭建生产环境时，你可以协同运维部门的同事一起完成，你可以将优化的思路告诉他们，由他们来完成操作会更加合适。

CPU 模式的优化

所谓 CPU 模式的调整，就是调整 CPU 的工作频率，使其呈现出不同的性能表现，以满足特定的业务使用场景。我们一般使用的 Linux 系统，也都有多种模式可供选择，像 PowerSave、OnDemand、Interactive、Performance 等，每个模式的调频方式都不同。

因为考虑到秒杀业务的特殊性，并且有时候活动非常的火热，但过段时间可能就降温了些，所以我们采用的模式也不相同。

像大促期间或者某段时间部分商品持续大力度营销，这时的活动非常火热，流量也高，所以我们需要将 CPU 模式调整成 Performance，即高性能模式。这时 CPU 一直处于超频状态，当然这种状态也是比较耗电的，但是为了更好地开展活动，还是需要打开的。

而当活动处于日常化时，此时流量较大促有很大差异，并且每天的流量相对稳定，这时候我们就可以将 CPU 模式切回成 PowerSave 模式，即节能模式，或者是切回系统的默认模式 OnDemand。这样的话，可以兼顾性能与资源开销，高性价比地支持秒杀活动。

网卡中断优化

那调完了 CPU，另一个需要优化的点就是网卡中断了。

“中断”是机器硬件与 CPU 交互的一种方式，即硬件告诉 CPU 有事情要处理了。而网卡中断，就是机器网卡告诉 CPU 要处理网络数据了。

前面我们就有说过秒杀的瞬时流量非常高，带来的问题就是一下子会有非常多的网络请求进来。网卡在收到网络信号后，会通知 CPU 来处理，这时如果我们没有调整过相关配置，那么很有可能处理网卡中断的 CPU 都集中在一个核上。

如果这个时候该 CPU 也在承担处理应用进程的任务，那么就有可能出现单核 CPU 飙升的问题，同时网络数据的处理也会受到影响，导致大量 TCP 重传现象的发生。所以这个时候，我们要做的就是合理分配多核 CPU 资源，专门拿出一个核来处理网卡中断。

操作的过程大致可以分成 3 步：

1. 查看在流量高峰时，是否处理网卡中断的工作都集中在同一个核上；
2. 找到网卡中断的 IRQ（硬件设备的一个编号，让 CPU 知道是哪个硬件的中断信号）；
3. 将网卡的 IRQ 与一个特定的 CPU 核进行绑定。

我们这么做的目的，其实就是在多核 CPU 下，让一个进程在某个给定的 CPU 上尽量长时间地运行而不被迁移到其他处理器。这样做的好处就是：一方面可以减少 CPU 调度产生的开销；另一方面可以提高每个 CPU 核的缓存命中率。

同样地，如果我们的 Nginx 服务和我们的 Redis 也安装在同一台物理机上，那么我们也可以将 Redis 进程以及 Nginx 进程分别绑定到不同的核上。比如我们有 16 核，那这个时候，我们可以将其中的 10 核绑定 Nginx 进程，2 核用来绑定 Redis 实例，1 核用来绑定处理网卡中断，剩下的可以选择再开 Nginx 实例或者给其他进程使用，这样每个核都可以专注处理自己的进程任务，不用切换拷贝内存等，从而大大提高了整体的服务响应性能。

以上 Redis 的绑核操作可以通过 taskset 来完成，而 Nginx 的绑核则可以通过 Nginx 的配置来直接绑定。那么接下来我们就一起看下 Nginx 配置方面的优化点。

Nginx 配置优化

前面在介绍 Nginx 的时候，我有给你展示过这张 Nginx 配置的模块结构图，每个模块都可以做相应的配置。



nginx.conf 配置



下面我们就按照顺序，依次介绍下各模块中重要配置的优化方式，并在最后附上一个最终建议优化配置。

全局模块配置

首先是全局模块配置 `worker_processes`，即用来处理网络请求的工作进程数。这个配置参数的设置非常关键，它直接会影响到整个 Nginx 服务的吞吐量，设置小了，发挥不了服务器的硬件水平，设置过大，还会起到反效果。

那设置多少合适呢？这个就要看我们机器的硬件配置了，我们建议工作进程数和 CPU 核数保持一致，这是种比较理想的状态。但就像上面提到的，如果机器上还部署了其他应用，像 Redis 实例等，那这个时候就要考虑到其他应用对 CPU 资源的占用，并且需要结合绑核

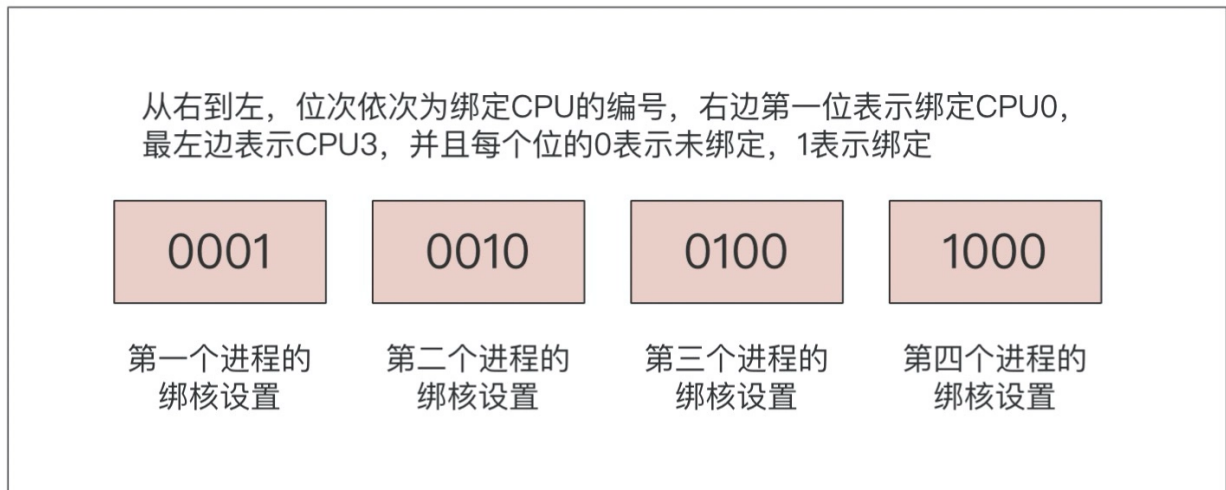
操作，尽量让一个 CPU 核能专门处理一个工作进程。所以我们下面结合绑核配置一起看下。

worker_cpu_affinity：绑核的目的，上面也已经介绍过了。当我们了解了机器的配置，以及部署在机器的应用以后，我们就可以合理地分配 CPU 资源，以 4 核 CPU 绑核为例，其绑定语法如下：

```
1 worker_cpu_affinity 0001 0010 0100 1000;
```

[复制代码](#)

针对语法的说明可参考下图：

 极客时间

通过合理地设置工作进程，并将工作进程与 CPU 一一绑定，可以有效利用机器资源，提高服务的吞吐量与整体响应性能。

那说到吞吐量呢，还得介绍一个关键的配置参数，即工作进程可以打开的最大文件描述符数量，其配置指令为 **worker_rlimit_nofile**。

worker_rlimit_nofile：我们都知道网络通信的底层是通过 socket 来建立连接的，而每一个 socket 又会打开一个文件描述符，所以文件描述符的数量设置会影响服务器处理网络连接的上限。如果设置过小，那么超过文件描述符数量的连接都会被直接返回。而该配置又

和单个工作进程可以建立的最大连接数量息息相关，所以我们和下面的 `worker_connections` 一起说下。

events 模块配置

`worker_connections` 指令是在 `events` 模块配置中使用的。

worker_connections：刚上面提到，单个工作进程可以建立的最大连接数量，是受 `worker_rlimit_nofile` 配置限制的，理论上该值的设置应等于最大文件描述符数量除以工作进程数，但因为工作进程处理请求并不是均匀的，所以将该值的设置只要小于等于最大文件描述符数量即可。一般在线上使用时，我们会将这两个的配置值都设置为 65535。

那既然说到 `event` 模块的配置，那我们就再说几个其他的常用配置。

accept_mutex：这个指令是用来配置工作进程接受新连接的方式。如果开启，那么工作进程会轮流接受新的连接，即采用互斥锁的方式。否则的话，当有新连接进来之后，所有的工作进程都会被唤醒，但只有一个可以接受新连接并处理。其他进程如果没有连接处理，则过段时间会继续进入等待状态，而这个时间段内对 CPU 资源是有消耗的。由此可见，如果我们流量较小时，建议打开，相反，则建议关闭。如果该配置启用，那么最好也设置一下以下配置。

accept_mutex_delay：该指令是配合 `accept_mutex` 来使用，是设置工作进程取得互斥锁后接受新连接的超时时间。超过设置时间，其他进程将可以获得互斥锁，这样可以防止上个进程拿到锁后一直不释放，导致处理请求受阻。

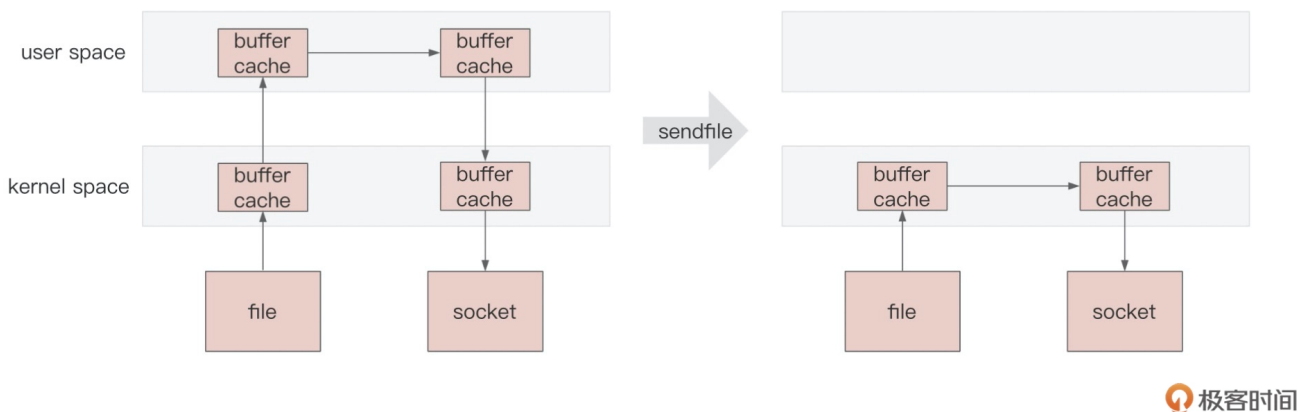
HTTP 模块配置

那 `events` 配置模块差不多就这些了，下面我们开始介绍 HTTP 模块的配置。这块都是关于 HTTP 请求处理相关的设置，比较多，我们这里会总结出一些针对秒杀场景的常用优化项。

sendfile：这个是操作系统用来优化文件传输提供的一个函数。

正常的文件传输，读时需要将数据文件从硬盘拷到内核空间，再从内核空间拷贝到用户空间，写时再依次拷贝出，同时也伴随着上下文的切换。

而 `sendfile` 的做法是省略掉了内核空间和用户空间的拷贝以及上下文切换操作，这也叫做零拷贝，节省资源的同时提高了效率，具体如下图所示：



因为秒杀有文件传输的场景，所以建议打开这个配置（通过 `upstream` 的文件传输，是不受该配置影响的）。打开该配置后，就可以继续使用另一个指令 `tcp_nopush` 了。

`tcp_nopush`：该配置只有在打开 `sendfile` 配置的情况下才能生效。

简单来说，该配置是关于 TCP 传输的配置。如果打开了 `nopush`，那么在 Nginx 响应客户端请求时，会优化响应数据包的发送模式，即将多个较小的数据包合并发送，就像响应头和响应体。这样做的好处是可以减少网络拥堵，优化网络传输，当然也会牺牲稍许实时性的体验。

那么说到这，就不得不说下另一个关于 TCP 传输的配置了。

`tcp_nodelay`：顾名思义，如果开启，就是用来降低网络延时的。

其做法和上面的 `tcp_nopush` 相反，追求数据包的实时传输，对于秒杀这种网络负载较高的场景，一般不推荐打开，并且该配置只在长连接条件下才能生效，而秒杀结算页的 HTTP 请求一般都使用短连接（这里以及下文提到的长、短连接，都是指 TCP 层面的）。那在 Nginx 上也可以通过指令来配置客户端的连接处理方式，即 `keepalive_timeout`。

`keepalive_timeout`：一般 HTTP 请求是要使用长连接还是短连接，需要客户端和服务端都支持。

客户端在使用 HTTP1.1 之后的协议版本，默认都是长连接，如下图所示：

Name	Status	Protocol	Type	Initiator	Size	Time	Waterfall
☐ getInterBySlug	200	h2	xhr	sdk.js:1	368 B	190 ms	
☒ init.action	200	h2	xhr	sdk.js:1	2.0 kB	233 ms	

×

Headers

Preview

Response

Initiator

Timing

▼ Request Headers

View source

Accept: application/json, text/plain, */*

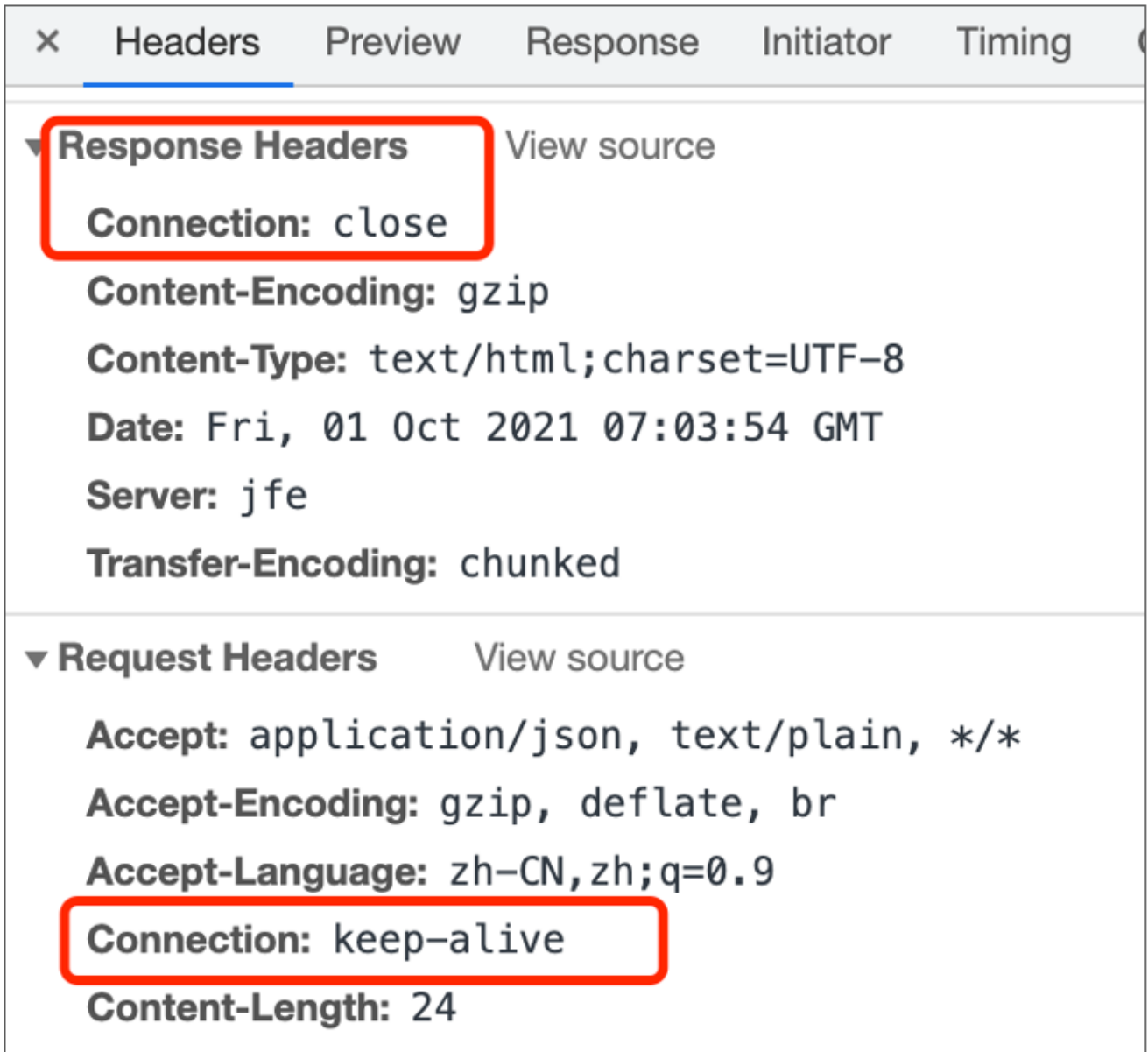
Accept-Encoding: gzip, deflate, br

Accept-Language: zh-CN,zh;q=0.9

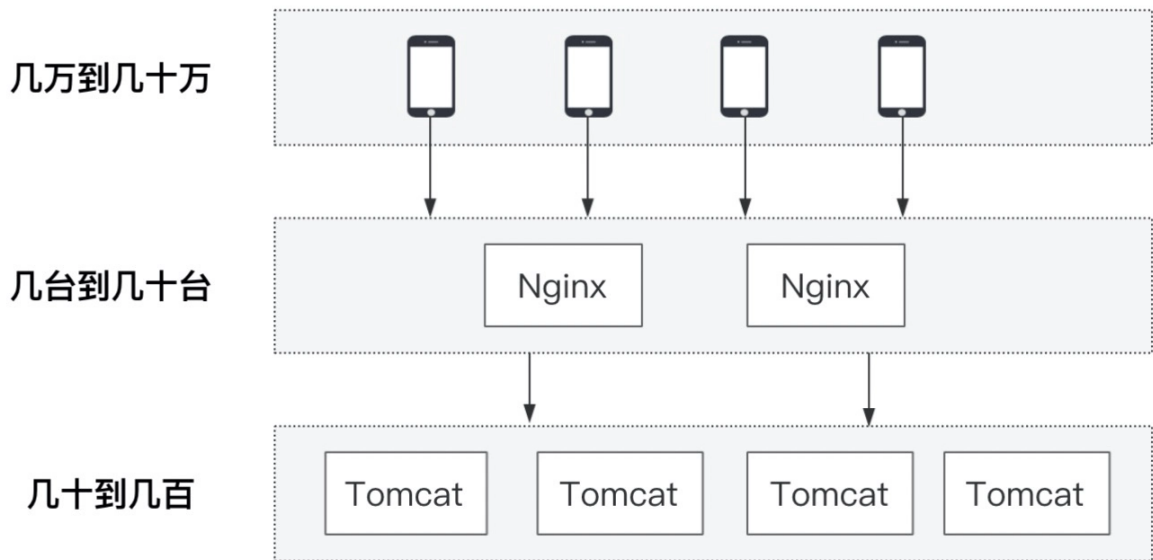
Connection: keep-alive

Content-Length: 24

协议版本是 HTTP2.0 的话，如果没有特殊配置，请求头里会有 keep-alive 的设置。但我们可以 Nginx 通过配置空闲连接的存活时间为 0，来关闭长连接，使其变成短连接。效果如下：



我们知道 Nginx 要连接的不仅有客户端，还有下游的 Web 服务。那针对上游客户端以及下游服务端，采用的连接方式会有什么不同呢？我们看下这张图：



Nginx 之所以针对客户端采用短连接，是因为客户端数量太多了，且交互不频繁。如果都用长连接，那么很快服务端连接将会被占完。

但对于下游，Nginx 相对就成了客户端了，其数量只有几台到几十台之间不等，且与后端 Tomcat 交互是十分频繁的，如果不停地创建连接，将会造成非常大的性能损耗，所以最好采用长连接的方式。长连接配置的方式，就放到最后的汇总配置里了。

最后再说几个与下游 Web 服务相关的超时配置。

它们分别是 `proxy_connect_timeout`、`proxy_send_timeout`、`proxy_read_timeout`，即连接建立超时时间、发送请求超时时间、读取响应超时时间。

如果不合理设置这些超时时间，就会因为各种网络状况导致 Nginx 与 Web 服务之间数据传输受阻，客户端将会一直等待，体验极差。所以我们需要根据接口的正常响应时间，设置一个合理的超时时间，等待超过超时配置，再将返回提示给到用户。具体的设置我也放到汇总配置里了。

总结

这节课我们介绍了物理机与 Nginx 相关配置的优化，优化的方向无非是对内存、CPU、IO（磁盘 IO 和网络 IO）的优化。

所以对于物理机，我们主要从调整 CPU 的工作模式来入手，根据不同的秒杀业务场景，我们切换不同的工作模式。像大促期间或者活动非常火热时，我们将 CPU 模式切换到高性能模式，以得到更好的响应性能，当然代价就是耗电增加；如果活动比较平稳，我们可以考虑切换到省电模式，这样可以节约成本。

另外在秒杀高峰期间，网络负载重，TCP 重传现象频发，我们也做了针对性的措施，即通过绑定专门 CPU 来处理网卡中断。当然绑核的操作不只可以针对网卡中断，还可以绑定 Nginx 进程以及部署的其他应用服务。这么做的目的，一方面是为了减少 CPU 调度产生的开销，另一方面也可以提高每个 CPU 核的缓存命中率。

之后我们又讲到了 Nginx 的优化，分别针对客户端以及下游服务端，从网络的连接、传输、超时等方面做了不同的配置讲解。具体的 Nginx 优化配置你可以参考以下示例。

nginx.conf 配置如下：

 复制代码

```
1  #工作进程：根据CPU核数以及机器实际部署项目来定，建议小于等于实际可使用CPU核数
2  worker_processes 2;
3
4  #绑核：MacOS不支持。
5  #worker_cpu_affinity    01 10;
6
7  #工作进程可打开的最大文件描述符数量，建议65535
8  worker_rlimit_nofile 65535;
9
10 #日志：路径与打印级别
11 error_log logs/error.log error;
12
13
14
15 events {
16     #指定处理连接的方法，可以不设置，默认会根据平台选最高效的方法，比如Linux是epoll
17     #use epoll;
18     #一个工作进程的最大连接数：默认512，建议小于等于worker_rlimit_nofile
19     worker_connections 65535;
20     #工作进程接受请求互斥，默认值off,如果流量较低，可以设置为on
21     #accept_mutex off;
22     #accept_mutex_delay 50ms;
23 }
24
25 http {
26     #关闭非延时设置
27     tcp_nodelay off;
28     #优化文件传输效率
```

```
29     sendfile      on;
30     #降低网络堵塞
31     tcp_nopush    on;
32
33     #与客户端使用短连接
34     keepalive_timeout 0;
35     #与下游服务使用长连接,指定HTTP协议版本,并清除header中的Connection,默认是close
36     proxy_http_version 1.1;
37     proxy_set_header Connection "";
38
39     #将客户端IP放在header里传给下游,不然下游获取不到客户端真实IP
40     proxy_set_header X-Real-IP $remote_addr;
41
42     #与下游服务的连接建立超时时间
43     proxy_connect_timeout 500ms;
44     #向下游服务发送数据超时时间
45     proxy_send_timeout 500ms;
46     #从下游服务拿到响应结果的超时时间(可以简单理解成Nginx多长时间内,拿不到响应结果,就
47     #这个根据每个接口的响应性能不同,可以在每个location单独设置
48     proxy_read_timeout 3000ms;
49
50     #开启响应结果的压缩
51     gzip on;
52     #压缩的最小长度,小于该配置的不压缩
53     gzip_min_length 1k;
54     #执行压缩的缓存区数量以及大小,可以使用默认配置,根据平台自动变化
55     gzip_buffers 4 8k;
56     #执行压缩的HTTP请求的最低协议版本,可以不设置,默认就是1.1
57     gzip_http_version 1.1;
58     #哪些响应类型,会执行压缩,如果静态资源放到CDN了,那这里只要配置文本和html即可
59     gzip_types      text/plain;
60
61
62     #access_log的日志格式
63     log_format access '$remote_addr - $remote_user [$time_local] "$request
64     "$upstream_addr" $upstream_status" $upstream_response_time" upstream
65
66     #加载lua文件
67     lua_package_path "/Users/~/.Documents/seckillproject/demo-nginx/lua/?.lua;
68     #导入其他文件
69     include /Users/~/.Documents/seckillproject/demo-nginx/domain/domain.conf;
70     include /Users/~/.Documents/seckillproject/demo-nginx/domain/internal.conf;
71     include /Users/~/.Documents/seckillproject/demo-nginx/config/upstream.conf;
72     include /Users/~/.Documents/seckillproject/demo-nginx/config/common.conf;
73 }
```

当然,今天说到的这些优化方向和优化点,考虑到机器配置、网络环境、业务功能等因素的差异,我们还需要依据实际压测效果来做灵活的调整,但总体优化思路是不变的。

思考题

以上针对 Nginx 的优化配置有很多的指令，这些指令可以在不同的模块进行配置，比如像 proxy_set_header 这种，既可以在 HTTP 全局模块配置，也可以在 server 模块进行配置，那类似这样的指令设置，都会考虑哪些因素呢？

期待你的思考，欢迎在评论区中和我讨论问题，交流经验！我们下节课再见。

分享给需要的人，Ta 订阅后你可得 **20 元现金奖励**

生成海报并分享

赞 1

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 10 | 不差毫厘：秒杀的库存与限购

下一篇 12 | 高性能优化：单机Java极致优化

1024 活动特惠

VIP 年卡直降 ¥2000

新课上线即解锁，享 365 天畅看全场

超值拿下 ¥999



精选留言 (1)

写留言



呆萌白的大白。

2021-10-21

sendfile on;
tcp_nopush on;
tcp_nodelay on;
这三个选项一般同时开启，并不冲突。
tcp_nopush on保证数据包的长度尽量到mss...

展开

